Lecture 10
Topics in Configuration Managements
1. Componentization
2. Product-line family
Last lecture …

1. Sign a contract
2. Design by contract
   *Three kinds of design contracts*
3. Programming by contract
   *Three kinds of programming practices by contract*
Today …

1. Problems in legacy software development
2. Componentization
   1. Redundancy removal
   2. Header Restructuring
   3. Clustering (repackaging)
3. Feature oriented programming
4. Summary
1. Problems facing SE

- Software are getting more complex
  - Code size getting larger, more dependencies
  - More developers are involved
  - More users and stakeholders
  - Understandability, productivity are dropping
- Thus, Control the complexity is the central theme of software engineering
- How to improve so that people can develop in parallel and incrementally? Sync-and-Stabilize or “Daily build” approach
- Componentization and Software Product-line family are good solutions to the problem
2. Components

• Modules have high **cohesion** and low **coupling**
• To support parallel development, ideally, components can be **independently** compiled and tested
• A component has an **interface** (set of operations) through which other components can interact
• A web service is a component that has a **standardized** interface and **interoperability** regardless of programming languages
Legacy software

• Legacy software typically contains large set of program files, but not well modularized
• Redundancy: the interfaces of “components” in legacy software are bloated
  – A prolonged fresh build time
• False dependencies: including unnecessary program units for the component
  – Too complex to be understood
  – A prolonged incremental build time
• We will show C/C++ as an example, but the problem exists for other PL as well
Example 1. Hello world

```c
#include <stdio.h>
void main () {
    printf ("Hello, world!");
}
```

- **How many LOC after inclusion?** 767
  
gcc -E -P hello.c -o hello.o
  wc hello.o

- **How many LOC is needed?** 4
  
gcc -E -P -fdump-program-unit hello.c

- **The `#include` shall expand to a single line:**
  
  ```c
  int __attribute__((__cdecl__)) printf( const char*, ...);
  ```
2.1 Componentization

- Restructuring by removing unnecessary units in the program
- A restructuring unit is a statement \textit{declaring}, or a \textit{defining} of the user-defined symbols, such as \textit{functions}, \textit{variables}, \textit{classes}, \textit{structures}, \textit{types}, etc.
- A local variable, parameter, a field or a method of the class are not considered as a restructuring unit because removing them may affect the semantic of the program
- What is the difference between declaration and definition? Declaration can occur multiple times, definition can only occur once.
- Preserving semantics: (1) maintain the dependencies such that compiler won’t complain about undefined symbols; (2) make sure necessary definitions are kept in the compilation units
2.2 Redundancy removal

• As shown in previous example, redundancy happens when some program declaration are unnecessary

• How to tell this?

• In GCC 3.4.0, we change its parser such that a symbol transitively dependent by the definitions will be kept in the precompiled program

• Very efficient and beneficial compilation time + precompilation time < original compilation time
Example 2. Removing redundancies along parsing

1. typedef int NUMBER; //PU01
2. struct node; //PU02 forward:node@2
3. typedef struct node { //PU03 type:list@3
4.   float value; // struct:node@3
5.   struct node* next; // <- PU03, PU02
6. } *list; //
7. struct A { //PU04 struct:A
8.   union {
9.     NUMBER value; // <- PU01
10. } u;
11. };
12. extern int //
13. printf(char *format,...); //PU05 funcdcl:printf@5
14. enum { //PU06 enum:<anonymous>@6
15.   Satisfied, // enumerator:Satisfied@6
16.   Denied, // enumerator:Denied@6
17. };
18. int main(argc, argv) //PU07 funcdef:main@7
19. int argc; char **argv; //
20. {
21.   list l, n; // <- PU03
22.   for (n = l; n; n=n->next) //
23.     printf("f", n->value); // <- PU05
24.   return (int) Satisfied; // <- PU06
25. }
//
2.3 Header restructuring

• Configuration management: to maintain the software when changes happen
  For example: CVS
• Removing redundancies in the preprocessed program does not solve the problem for
  incremental changes
• A compilation unit does not need to recompile when its dependent symbols are not changed at
  all
• Such unnecessary recompilations are caused by false dependencies
Example 3. False dependency

```c
void foo();
void bar();

#include "foo.h"
int main() {
    foo();
}
```
The removal of false dependencies

- Identify dependencies
- Partition the definition and declaration units into separate files, replacing dependencies with “#include”
- Grouping the declarations into larger headers, if it does not incur false dependency
- The code generation process can be done efficiently
(a) Program unit sequences after redundancy removal where \( h_i \) is the i-th global declaration and \( C_j \) is the sequence of definitions in the j-th compilation unit.

\[
\begin{align*}
H_0 &= <h3> \\
\{C_1, C_2, C_3\} \\
H_1 &= <h1, h2> \\
\{C_1, C_2\} \\
H_2 &= <h4> \\
\{C_3\} \\
\text{()} &
\end{align*}
\]

(b) The implicit light-weight PUDG

(c) The partitioning lattice

(d) Generating ordered header inclusions
2.4 Clustering

- Problem: too many headers are generated, because we get rid of all false dependencies.
- Tradeoff: Can we tolerate some false dependency for smaller number of headers, that is, to group them further into larger files?
- Clustering is to group related things together, the technique is often used in data mining and machine learning.
- We want to cluster generated headers use the hints of dependencies.
LIMBO clustering

• LIMBO is a clustering technique to minimizing information loss in dependency graphs
• Group A, B into a cluster does not have information loss if both depends on same entities, e.g.
  A depends on A1, A2
  B depends on A1, A2
• Group A, B into a cluster has information loss if they depends on different entities, e.g.
  A depends on A1, A2
  B depends on B1, B2
• The idea is to quantify the information loss and rank them so that minimal loss is the priority
Example 4. VIM 6.2

- We have removed around 70% redundancies in LOC
- We have removed all false dependencies, which generates 952 headers
- Using dependencies and the LIMBO clustering, we got only 3 clusters (corresponds to the MVC architectural pattern) and 5 headers
Experiments: fresh build time

![Bar chart showing compilation time for different build options]

- Original
- Precompiled
- Restructured
Experiments: fresh build speedups
Experiment: incremental build time

estimated incremental recompilation time

- original
- precompiled
- restructured
- componentized

Second

compilation units

Spring 2005
ECE450H1S
Software Engineering II
2.5 More code removal?

- **Dead code elimination**
  ```java
  int add(int x, int y) {
      int r1 = x + y;
      int r2 = x * y;
      return r1;
  }
  ```

- **Unused fields and methods**
  ```java
  class A {
      double value;
      int getValue() { return value; }
      public static void main(String args[]) {
          printf(“Hello world!”);
      }
  }
  ```
3. Variability in Product-line Family

- Consider Daimler Chrislrer (car manufacturer), every product out of the product-line is different from each other — [Czarnecki]
- Why? Because the Factory produces software that variability in every feature of the car
- Can we do the same in software industry? SAP’s approach: Domain engineering
- Feature models capture variability in the solution space, whereas goal models capture variability in the problem space
3.1 Feature model

CaptainFeature is a feature modeling tool [Czarnecki]
A feature is either Mandatory, Optional, Alternative or (Inclusive) Or.
Example from Batory’s tutorial

4x4x2 variants
Software Feature Model

- A software system is composed of features
- Features can be organized in a hierarchy
- Example
  - eclipse/features/feature.xml
  - eclipse/plugings/plugin.xml...
3.2 Feature oriented programming

- Supported by the AHEAD tool suite
- Key idea is to represent a feature as a layer of the incremental pieces of modules
  - In Hyper/J, this is called “concern graph”
  - In AspectJ, it is called aspect crosscutting
- FOP versus AOP?
Example

class A {
    data1; method1;
    data2; method2;
    data3; method3;
};

class A {};  ...Core prg. as a constant c
class A { data1; method1; }; ...Feature as a function i
class A { data2; method2; }; ...Feature as a function j
class A { data3; method3; }; ...Feature as a function k

• Mixing them k(j(i(c)))
• Advantages:
  Incremental and parallel development
  Step-wise refinement
• Risk:
  How to guarantee the semantics and information hiding?
3.3 Generative programming

- Templates in C++: stack<int>
- Templates in code generators (Eclipse) 
  Generating class, method, test cases, etc.
- Generated code in the Visual programming 
  Visual Studio, Visual Editor, etc. Generating GUI code
- What else does generative programming do? Derives a 
  configuration from the feature model. Each configuration 
  leads to one variant of the product
  - #if engine==GASOLINE
    
    ...
    
    #endif
  - -Dengine=GASOLINE
  - CaptainFeature -> Configuration (XML)
- You may apply the variability configuration at compile-
  time, deploy-time, run-time
3.4 Industrial practice: Partial classes

- .NET framework 2.0 (ASP.NET magazine)
- Implemented in the CLR: C#, C++, VB
- Proposed to solve problem for mixing generated code (visual programming) and user code
- Now a class definition can scatter over multiple files as long as there is a “partial” modifier

```csharp
partial class A { data1; method1; };
partial class A { data2; method2; };
partial class A { data3; method3; };
```
- The weaving is done by the .NET compiler
4. Your exercise

• Consider componentization of your modules: minimize the interface
• Each component is a module that implements part of a feature, they can be organized into a (layered) feature model, and converting the program into a set of features (FOP)
• Create a feature model to show the distinctiveness of your product over other teams? ----- bonus J
• Use feature model to know whether you can produce a generic software as a product line family, to integrate with other team’s various products
5. Summary

• Why componentization is important?
• How can you turn legacy software into components?
• How can you decompose components into features and assemble them back?
• What’s the relation among CBSE (COTS), FOP and AOP?
Further readings


• Y. Yu, H. Dayani-Fard, J. Mylopoulos. “Remove false code dependencies to speedup up build process”, CASCON’03.